

# Shape Recognition and Corner Points Detection in 2D Drawings Using a Machine Learning Long Short-Term Memory (LSTM) Approach


Zahra Karimi<sup>1</sup>, Shrikant Savant<sup>2</sup>, Abe Zeid<sup>1,\*</sup>, and Sagar Kamarthi<sup>1</sup>

## ABSTRACT

Creating a 2D geometry model from an image poses challenges for CAD users due to factors such as noise, segmentation difficulties, complex geometric structures, scale and perspective variations, and the need for CAD system compatibility. In this paper, we propose a novel deep learning approach utilizing Long-Short Term Memory (LSTM) to address these challenges. Our approach decomposes the shapes in the images into line and curve segments and accurately locates their intersection points. To enhance the model's performance, we introduce two distinct types of features (angle and curvature features) and optimize the model through hyperparameter tuning. The resulting model exhibits robustness against noise, varying image sizes, and can effectively locate different types of intersection points. To evaluate the proposed model, we have developed a Python-based software and conducted experiments on a dataset comprising of 200 shapes with seven different resolutions. Comparative analysis against a state-of-the-art method (TCVD) from the literature demonstrates that our approach achieves higher accuracy in terms of line, curve, and intersection point detection.

Submitted: January 02, 2024

Published: March 05, 2024

 10.24018/ejai.2024.3.1.34

<sup>1</sup>Department of Mechanical and Industrial Engineering, Northeastern University, United States.

<sup>2</sup>Dassault Systèmes, United States.

\*Corresponding Author:  
e-mail: zeid@coe.neu.edu

**Keywords:** AI, CAD, ML, Shape recognition.

## 1. INTRODUCTION

Sketching plays a vital role in the design culture and is an integral part of the design process [1]. Numerous powerful commercial CAD tools are available for both 2D and 3D design. However, creating geometry from existing data sources can be time-consuming. Often, users face the need to construct models based on 2D drawings, PDFs, photos, or non-native and neutral format CAD files. This requirement typically arises when they are unable to obtain a high-quality model of a component for their design or when they already possess a legacy software package and wish to reuse and modify it in a new environment while maintaining its feature-based characteristics.

CAD users who contemplate switching between different CAD software often worry about leaving their legacy data behind. The fear of having to remodel everything in a new environment is understandable and a common concern in the CAD practice. The conversion of legacy data is viewed as a significant obstacle to changing software packages, particularly when there is a need to continue

ongoing projects or incorporate components from previous designs into new ones, requiring modifications to fit both old and new designs. Dealing with legacy products that may lack available drawings further complicates matters, leaving customers with no choice but to provide a picture and a few measurements to create replacements or upgrades.

Given the crucial role of sketches in the design process, it is highly valuable to optimize sketch creation methods to expedite the overall design process. By providing a tool that can analyze an image and extract the contained information to construct 2D sketches, we can significantly accelerate the concept-to-design workflow.

The detection of corner and tangent points is an active research area within stroke segmentation [2]. Several studies have been conducted to explore techniques for segmenting hand-drawn pen strokes into lines and arcs, with a particular focus on corner detection. Numerous techniques have been developed specifically to address the challenges associated with accurately identifying and locating corners in such strokes [3]–[5].



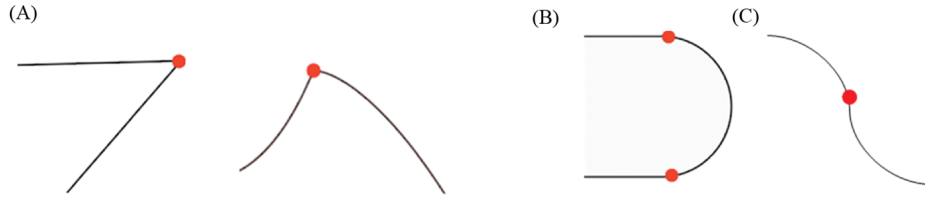


Fig. 1. Different types of corners: (A) Intersection of two lines, (B) Line intersects with an arc, (C) Inflection point of two edges.

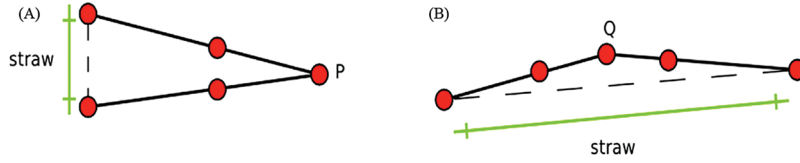


Fig. 2. Straw method: (A) Corner points (P) with short straw, (B) Non-corner point with long straw (Q).

A corner refers to the point where two edges intersect, such as the intersection of two lines within polylines (Fig. 1A). Additionally, a corner can also be a tangent point, where a line intersects with an arc (Fig. 1B). Another way to define a corner is as a point characterized by two distinct and prominent edge directions within its local neighborhood (Fig. 1C) [6].

There are two major approaches for Corner Detection: Traditional segmentation techniques, which typically rely on heuristic algorithms and empirically determined parameters, like curvature, and machine learning (ML)-based stroke segmentation methods, which use various shape features to train a model. The latter is general and extensible compared to heuristic-based approaches [7].

As an example of the first approach, Wolin *et al.* [8] has implemented Short-Straw, an effective and easy implementation technique for corner detection of polylines. They defined a parameter “Straw”, which calculates the euclidean distance between two neighbours of the target point:

$$Straw = |P_i - W, P_i + W| \quad (1)$$

where  $P$  is the target point, and  $W$  is a constant number. The points with minimum straw (i.e., less than a threshold) are identified as corner points (Fig. 2A), and other points above the threshold are labeled as non-corner points (Fig. 2B). This method only works for polylines and not for the strokes with arcs.

Xiong and LaViola [9] expanded this method and proposed IStraw method. It improves corner finding performance for strokes with arcs. In their approach, they added a new feature based on forming angles at each point. They observed that the angle increases in the presence of a false corner on a curve (Fig. 3B), while it remains unchanged for a genuine corner as the vertex (Fig. 3A).

Albert *et al.* [10] proposed the Tangent and Corner Vertices Detection (TCVD) method which uses parametric cubic curves of the stroke to calculate the radius function and then detect corner and tangent vertices. They have created software based on their approach that we use to compare to our proposed method [11].

For the ML-based corner point detection approach, Herold and Stahovich [12] proposed a machine learning

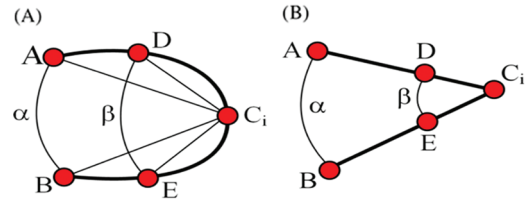


Fig. 3. IStraw method: (A) Non-corner point (curve point), (B) Corner point.

approach called ClassySeg for automatic strike segmentation. They used 48 geometrical features such as arc length, curvature, straw, and speed to train a decision tree classifier.

Long [13] proposed a deep learning approach using strike images as input, and points are classified as corner, line and curve using two residual neural networks. They showed that their deep learning model outperforms previous heuristic and empirical-based models.

In this study, we use a bi-directional long short-term memory (bi-LSTM) neural network to classify corners, lines, and arcs in the design images. 200 graphical shapes with 7 different sizes containing lines and arcs are created. The border points of the images are calculated using image processing techniques, and then the desired features are extracted. Then, we tested the bi-LSTM model using 10-fold cross-validation. Our results show that our bi-LSTM classifier outperforms existing ML-based classifiers.

## 2. CONCEPTS AND METHODS

### 2.1. Long Short-Term Memory (LSTM)

We use bi-directional long short-term memory (bi-LSTM) neural network to perform the classification and classify shapes into corners, lines, and arcs. LSTMs are types of Recurrent Neural Networks (RNNs) that use gating functions in their architecture [14]. Recurrent Neural Network (RNN) is a type of artificial neural network which saves the processing node and passes the information back to the model to improve the model performance. Each node in the RNN models acts as a memory cell, continuing the computation and implementation of the operation [15].

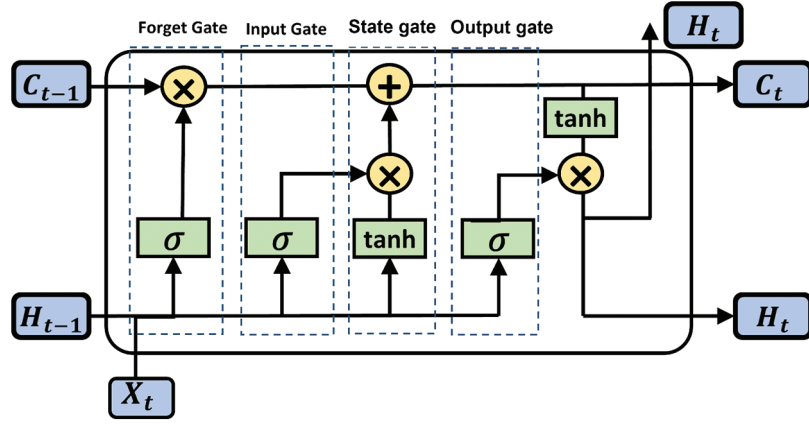


Fig. 4. LSTM structure layers, including the input gate, state gate, forget gate and output gates.

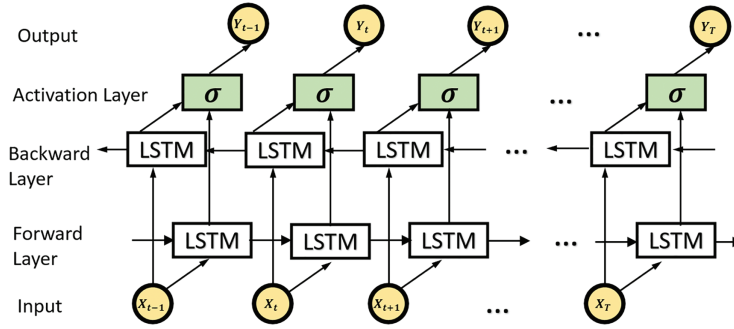


Fig. 5. Bi-LSTM structure schema.

The problem of vanishing gradients in ordinary RNNs is solved by LSTMs, which are capable of learning long-term dependencies. LSTMs use hidden layer units as memory cells which remember key information and forget unnecessary ones. Fig. 4 shows the flow of data series  $X_t$  to an LSTM layer.  $H_t$  and  $C_t$  are the hidden state and cell state at the time  $t$ , respectively. The flow of information in the LSTM unit is controlled by four gates: forget gate, input gate, state gate and output gate. The forget gate controls the level of forget from the previous hidden state; the input gate controls the level of cell input update; the cell state gate utilises the  $\tanh$  function on the previous hidden state, and the current input regulates the information flow within the network and output gate filters the amount of cell state added to the current hidden state.

$W$ ,  $R$  and  $b$  are input weights, recurrent weights and bias, respectively and are learnable weights of LSTM:

$$W = [W_i, W_f, W_c, W_o] \quad (2)$$

$$R = [R_i, R_f, R_c, R_o] \quad (3)$$

$$b = [b_i, b_f, b_c, b_o] \quad (4)$$

where  $i$ ,  $o$  and  $f$  represent input, output, and forget gate and  $c$  represent cell candidate. The input  $X(t)$  is mapped to the output  $H(t)$  by a series of equations:

$$i_t = \sigma_g(W_i x_t + R_i h_{t-1} + b_i) \quad (5)$$

$$f_t = \sigma_g(W_f x_t + R_f h_{t-1} + b_f) \quad (6)$$

$$g_t = \tanh(W_g x_t + R_g h_{t-1} + b_g) \quad (7)$$

$$o_t = \sigma_g(W_o x_t + R_o h_{t-1} + b_o) \quad (8)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (9)$$

$$h_t = o_t \odot \sigma_c(c_t) \quad (10)$$

where  $\tanh$  represents tangent hyperbolic function as the state activation function,  $\sigma$  represents the sigmoid function as the gate activation function and  $\odot$  represents the element-wise multiplication.

The original LSTM can only use data from previous steps. To overcome this drawback, the bi-LSTM has been introduced [16], which uses two separate LSTM for forward ( $H_{Forward}(t)$ ) and backward  $H_{Backward}(t)$  directions. Fig. 5 shows the bi-LSTM structure with both forward and backward directions. The final output of each hidden unit is calculated using both  $H_{Forward}(t)$  and  $H_{Backward}(t)$ :

$$H(t) = \Sigma(H_{Forward}(t) + H_{Backward}(t)) \quad (11)$$

## 2.2. Shapes and Labels

For this study, 200 shapes with different resolutions were created using Adobe Illustrator to train and test the proposed method. The shapes contain lines and arcs with different orientations and curvatures. All different types of corner points are introduced in the samples (Fig. 1), including a regular corner point created by the intersection

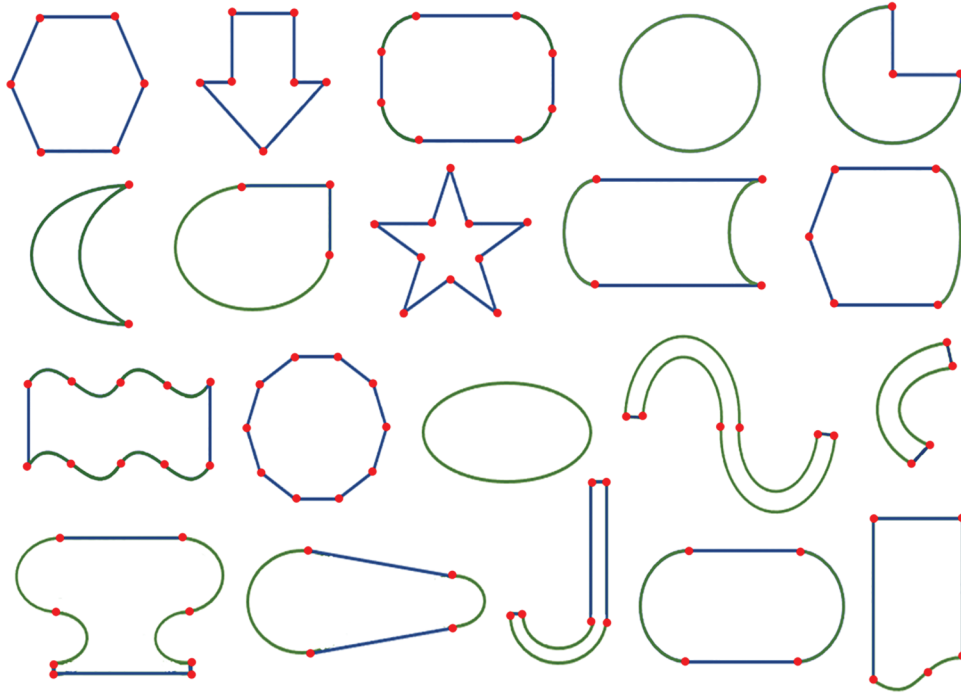


Fig. 6. Sample labelled training data set.

of two edges or lines, a tangent point created by the intersection of a line and an arc, and an inflection point where two dominant and different edge directions intersect in the local neighborhood of a point. 20 Samples of these shapes are shown in Fig. 6.

To study the LSTM model, a data set that includes 200 unique images with 7 different sizes has been used. Each image consists of a single shape. Each shape can include several strokes (lines and arcs). We used OpenCV library to extract the output contour of each shape. OpenCV (Open Source Computer Vision Library) is an open-source software library that provides a comprehensive set of tools and functions for computer vision and image processing tasks [17]. The contours are the pixel-wise border points sequence that will later be used to generate features for model training. Each point of the point sequence will be assigned a label of our 3 classes: corner point, line point and arc point. These types represent, respectively, a point in the corner area of the shape, a point inside the line element and a point which is part of the arc element of the shape. Several neighbors on the right and left of the actual corner are labeled as corner points. The length of this neighborhood is determined based on the size of the shape. This is done to reduce the number of missed corners in the training.

### 2.3. Feature Generation and Training

The raw sequence of border points is not suitable to be used as features since they don't explicitly show the geometrical changes in the shape to represent different labels. Here, we used two different sets of features, namely angle and curvature features, which are introduced as follows.

To create both angle and curvature features,  $n$  number of points on the right and left side of the sample point  $P$  are chosen. Then, the desired properties are extracted from these sets of points.

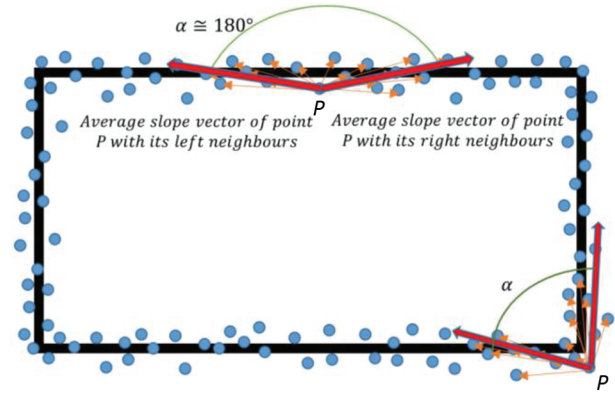


Fig. 7. Angle feature-angle between the average slope of right and left side.

For angle features, the vector direction from point  $P$  to neighboring points are created (Fig. 7) and then the vectors on the right and left side are summed, and unit vectors on each direction are calculated:

$$X_r = \frac{\sum (x_{ri})}{|\sum (x_{ri})|} \quad (12)$$

$$X_l = \frac{\sum (x_{li})}{|\sum (x_{li})|} \quad (13)$$

Then the angle feature between these two vectors is calculated as  $\alpha_n$ :

$$\alpha_n = \cos^{-1}(x_{nr} \cdot x_{nl}) \quad (14)$$

Fig. 8 shows angle features for a sample shape by using 10 neighbors to create the features. The angle feature pattern for each of three classes (corner segment, line segment, and curve segment) has unique characteristics. Fig. 8 shows sample corner segments (two red dots), line segments (edges labeled 1, 2, 3), and curve segments (arc

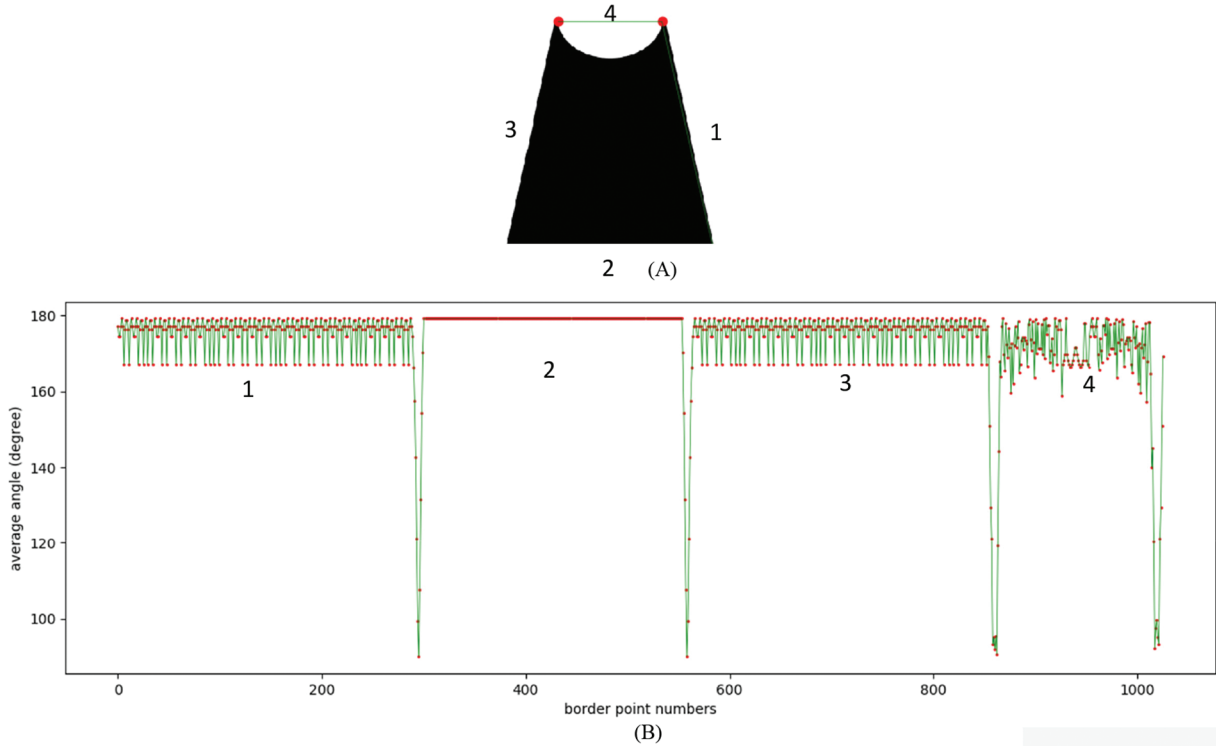


Fig. 8. Angle features: (A) Sample shape, (B) Angle feature pattern.

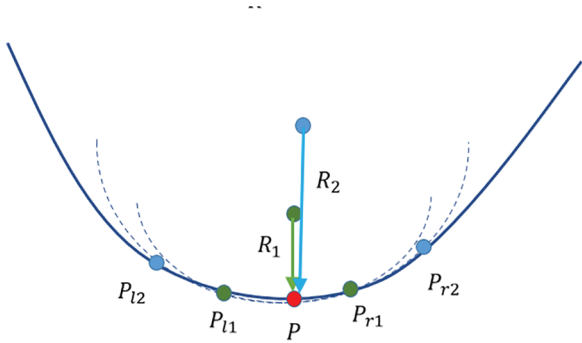


Fig. 9. Curvature feature.

labeled 4). There are big drops in angle pattern around the corners. The patterns for the line and curve features are distinguishable qualitatively.

We create curvature features as follows: for each point  $P$  on stroke, two neighbors by the distance  $n$  are chosen and a second order polynomial is fitted to these three points:

$$X = a_1 \times t^2 + b_1 \times t + c_1 \quad (15)$$

$$Y = a_2 \times t^2 + b_2 \times t + c_2 \quad (16)$$

Then the analytical curvature of the fitted polynomial is calculated. Fig. 9 shows fitted polynomial on a sample shape along with the radius of curvature for  $n = 1$  and  $n = 2$  neighbors.

Fig. 9 shows curvature features for  $n = 3$  neighbors. This figure shows that the curvature drops around the corners. The patterns for lines and curves are distinguishable and are different from the angle features, which helps the deep learning model identify the labels with higher accuracy.

#### 2.4. Training the Model

Fig. 10 shows the steps for tuning the model parameters. We first read image shapes and extract their border contours and then feed to our model referring procedure, which includes 3 steps:

1. Feature hyperparameters tuning includes numbers of neighbor counts and values for both the angle and curvature features,
2. Model hyperparameter tuning, which includes Bi-LSTM initial learning rate, numbers of hidden layers and different class weights,
3. And after training and during the post-processing, the region with less than specific numbers of certain classes will be marked, and the region will get a label of its neighbor of the left or right regions with these rules:

- For a line or curve candidate region (with a smaller size than a certain threshold), if the neighbor labels are different, the candidate region will be converted to a corner, but if the neighbor labels have the same label, the candidate region will be converted to the neighbor label.
- For a corner candidate region (with a smaller size than a certain threshold), if the neighbor labels are different, the candidate region will remain the corner, but if the neighbor regions have the same label, the candidate region will be converted to the neighbor label.

We added these two thresholds to our training process to find their optimum for each label. 162 different parameters combinations are identified. Then, using grid search and 10-fold cross-validation, the optimum set of parameters



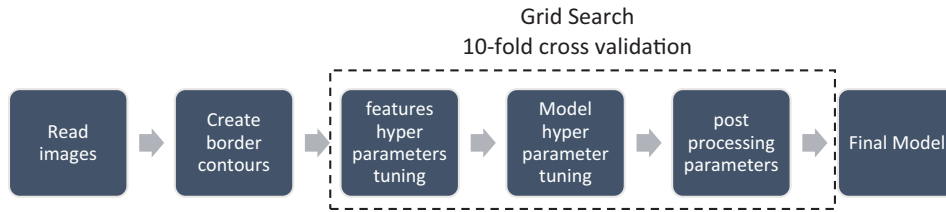


Fig. 10. Steps of model creation and hyper-parameters tuning.

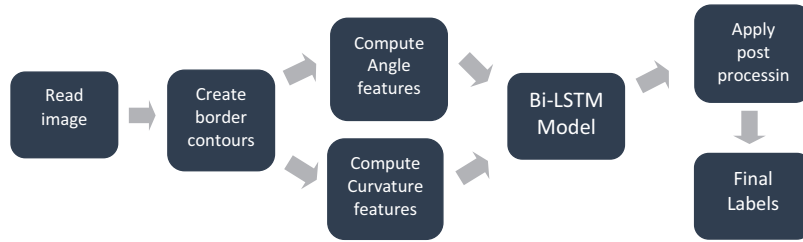


Fig. 11. Steps of model inference and classification of new images.

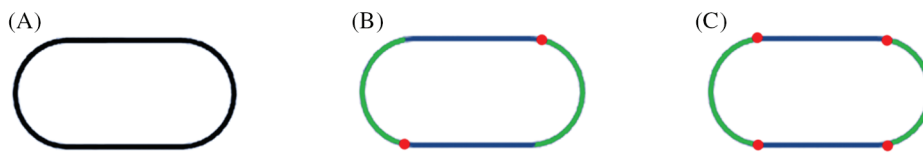


Fig. 12. Finding corner points: (A) Sample shape, (B) LSTM result, (C) Bi-LSTM result.

was discovered. Using these sets of parameters, the final model is established.

### 2.5. Model Inference

Fig. 11 shows the steps of model inference using the final trained model. First, we read the image and create its border contours. Then angle and curvature features are calculated, and those features will be fed into the trained Bi-LSTM model. We then apply the post processing rules to the candidate regions and final label will be generated.

## 3. RESULTS AND DISCUSSION

### 3.1. LSTM vs. Bi-LSTM Model

The original LSTM can only use data from previous steps. The initial results from the LSTM models show that there are plenty of tangent points that are missed. For improving tangent point detection performance, we used the Bi-LSTM model. The Bi-LSTM models show a great improvement in the case of tangent points detection (Fig. 12).

### 3.2. Angle vs. Curvature Features

To illustrate the difference between the angle feature and curvature feature, we choose a sample shape (Fig. 13) and the corresponding angle and curvature. Fig. 13 shows the example shape, and each segment of the shape is numbered from 1 to 6.

Fig. 13B shows the corresponding curvature feature, and Fig. 13C shows the angle feature. Segments 3 and 4 are zoomed in for better visibility. These figures show that curvature feature sign changes at inflection points (such as from segments 3 to 4 and 6 to 1) which makes it easier for the model to detect them. On the other hand, the angle

feature works better to detect corner points as it gives a higher drop in value on corner points, which are shown in corners from 1 to 2 and 4 to 5.

### 3.3. Effect of the Number of Neighbors on Feature Calculation

Here, we explore the effect of the number of neighbors in model classification. Fig. 14 shows a sample shape and Figs. 14B, 14C show the corresponding angle feature for three neighbors average and ten neighbor average, respectively. These figures show that low numbers of neighbors work better on small segments. For the three small line segments on top of the Fig. 14A, three neighbor averages (Fig. 14B) can distinguish the lines from each but for ten-neighbor average, the features for each region are diluted by other regions and make the three lines indistinguishable (Fig. 14C). On the other hand, features with high numbers of neighbors are more robust to noise and can better capture patterns on curves. The straight line in Fig. 14A has a step which appears to be a corner in three-neighbor average (Fig. 14B), but the step will be barely noticeable on ten-neighbor average (Fig. 14C).

### 3.4. Cross-Validation Experiment and Results

As mentioned in section 2.2, 200 shapes with 7 different resolutions are created and tested by our deep learning model. Our database contains 1174 corners, 794 lines and 379 curves. 150 shapes have been used in 10-fold cross validation to find the best hyper parameters and create the final model then the final model is applied on remaining 50 shapes. Tables I to III show that our deep learning results compare to Tangent and Corner Vertices Detection (TCVD) method [8]. TCVD method is a heuristic method

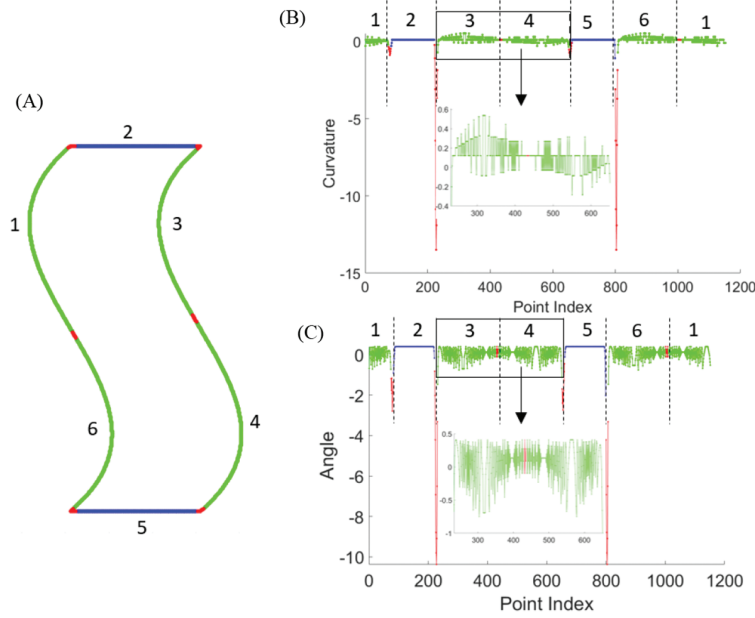


Fig. 13. Angle and curvature features: (A) Sample shape, (B) Curvature feature, (C) Angle feature.

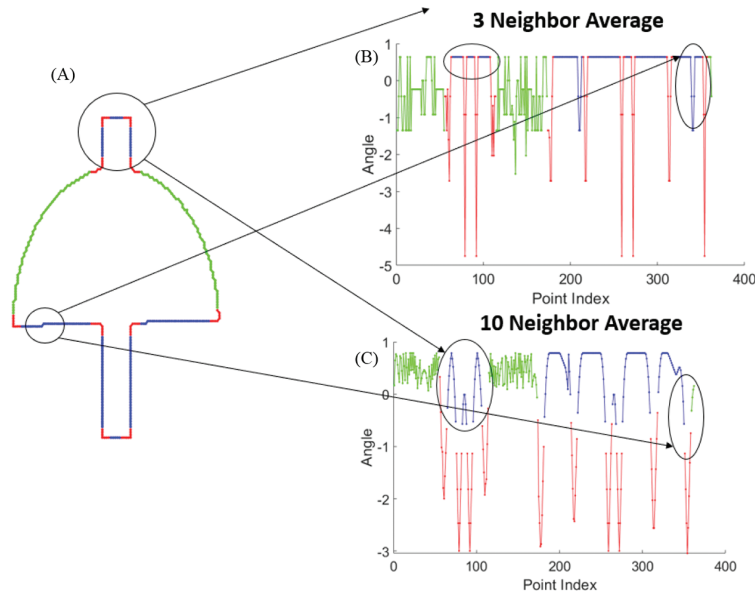


Fig. 14. Effect of number of neighbors in feature calculation: (A) Sample shape, (B) Angle feature for three-neighbors average, (C) Angle feature for ten-neighbors average.

TABLE I: SUMMARY OF ACCURACY OF TCVD MODEL VS. BI-LSTM MODEL FOR CORNER DETECTION

Accuracy results for corner detection	TCVD	Bi-LSTM
False positive corners	149	53
False negative corners	81	49
Correct corners found	1092	1124
Total corners	1173	1173
Correct corners accuracy	0.93	0.96

TABLE II: SUMMARY OF ACCURACY OF TCVD MODEL VS. BI-LSTM MODEL FOR LINES

Accuracy results for line detection	TCVD	Bi-LSTM
False positive lines	63	19
False negative lines	10	6
Correct lines found	784	788
Total lines	794	794
Correct lines accuracy	0.98	0.99

that uses the parametric cubic curvature approximation for stroke segmentation.

For evaluation purposes we define accuracy, false positive rate, and false negative rate as follows:

$$Accuracy = \frac{\text{Numbers of correct elements}}{\text{Total numbers of elements}} \quad (17)$$

$$False\ Positive\ Rate = \frac{\text{Numbers of incorrect elements}}{\text{Total numbers of elements}} \quad (18)$$

$$False\ Negative\ Rate = \frac{\text{Numbers of missed elements}}{\text{Total numbers of elements}} \quad (19)$$

When evaluating false positive corners, our Bi-LSTM algorithm demonstrated a reduction in the number of

TABLE III: SUMMARY OF ACCURACY OF TCVD MODEL VS. BI-LSTM MODEL FOR CURVES

Accuracy results for curve detection	TCVD	Bi-LSTM
False positive curves	54	25
False negative curves	57	24
Correct curves found	322	355
Total curves	379	379
Correct curve accuracy	0.84	0.93

incorrect corner detections compared to the TCVD algorithm (Table I). Specifically, TCVD reported 149 false positive corners, whereas our Bi-LSTM algorithm only had 53 false positive corners. Similarly, for false negative corners, our Bi-LSTM algorithm outperformed TCVD by producing fewer incorrect omissions. TCVD had 81 false negative corners, whereas our algorithm resulted in 49 false negative corners. Regarding the correct corners found, our Bi-LSTM algorithm detected more corners accurately than TCVD. Specifically, Bi-LSTM identified 1124 correct corners, while TCVD found 1092 correct corners. Considering the accuracy of correctly identified corners, our Bi-LSTM algorithm exhibited higher accuracy compared to TCVD. The correct corners accuracy for Bi-LSTM was 0.96, indicating that 96% of the corners detected by our algorithm were correct. TCVD had a slightly lower accuracy of 0.93, indicating that 93% of the corners detected by TCVD were accurate.

The accuracy results for line detection on 2D shapes using our proposed algorithm (Bi-LSTM) and TCVD are summarized in Table II. In terms of false positive lines, our Bi-LSTM algorithm outperformed TCVD by significantly reducing the number of incorrect line detections. TCVD reported 63 false positive lines, while our algorithm, Bi-LSTM, only had 19 false positive lines. Similarly, for false negative lines, our Bi-LSTM algorithm demonstrated better performance than TCVD by minimizing the number of missed line detections. TCVD had 10 false negative lines, while our algorithm resulted in only 6 false negative lines. Regarding the correct lines found, both TCVD and Bi-LSTM achieved high accuracy. TCVD detected 784 correct lines, while Bi-LSTM identified 788 correct lines. Considering the accuracy of correctly identified lines, both algorithms performed remarkably well. TCVD achieved a correct lines accuracy of 0.98, indicating that 98% of the lines detected by TCVD were accurate. Bi-LSTM exhibited a slightly higher accuracy of 0.99, indicating that 99% of the lines detected by our algorithm were correct.

Regarding false positive curves, our Bi-LSTM algorithm demonstrated better performance than TCVD by significantly reducing the number of incorrect curve detections (Table III). TCVD reported 54 false positive curves, while our algorithm, Bi-LSTM, only had 25 false positive curves. Similarly, for false negative curves, our Bi-LSTM algorithm outperformed TCVD by minimizing the number of missed curve detections. TCVD had 57 false negative curves, whereas our algorithm resulted in only 24 false negative curves. Regarding the correct curves found, our Bi-LSTM algorithm detected more curves accurately than TCVD. Specifically, Bi-LSTM identified 355 correct curves, while TCVD found 322 correct curves. Considering

the accuracy of correctly identified curves, our Bi-LSTM algorithm exhibited higher accuracy compared to TCVD. The correct curve accuracy for Bi-LSTM was 0.93, indicating that 93% of the curves detected by our algorithm were correct. TCVD had a lower accuracy of 0.84, indicating that 84% of the curves detected by TCVD were accurate.

The experimental findings indicate that Bi-LSTM exhibits better performance compared to TCVD when applied to the tested shapes. It is worth noting that TCVD was primarily designed and evaluated for handwritten strokes. In contrast, our Bi-LSTM model leverages two distinct types of features and incorporates varying numbers of neighbors, which effectively reduces the false positive rate. Additionally, by selecting Bi-LSTM as the deep learning model, which inherently captures both backward and forward information, we observe a notable decrease in the false negative rate, as demonstrated in section 3.1 of our study. These results emphasize the advantages of employing Bi-LSTM in curve detection and highlight its ability to surpass the performance of TCVD on the given shapes.

#### 4. CONCLUSION

In this paper, we propose a novel deep learning model that effectively breaks down CAD shapes into constructing line and curve segments while accurately identifying intersection points. Our approach introduces a sequential methodology fed into a bi-LSTM model, showcasing the strength of Bi-LSTM in shape segmentation. Leveraging the backward and featured capabilities of Bi-LSTM, along with the incorporation of two types of angle and curvature features, our model achieves the detection of various intersection point types.

To enhance the algorithm's robustness to noise and enable segmentation across different scales, we introduce the utilization of different neighbor sizes for feature calculation. The experimental results demonstrate the high accuracy of our model, achieving 96% accuracy in detecting intersection points and 97% accuracy in classifying line and curve segments. These results are compared against the heuristic TCVD model, highlighting the accuracy improvement our approach brings to shape segmentation.

The proposed algorithm utilizes machine learning techniques to detect and classify different shapes based on border points. This can be used in CAD software to automatically identify and label shapes within a design. It saves time and effort by automating the shape detection process, especially when dealing with complex designs that contain numerous shapes.

#### REFERENCES

- [1] Wolin A, Paulson B, Hammond T. Sort, merge, repeat: an algorithm for effectively finding corners in hand-sketched strokes. *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, 2009.
- [2] Chen J, Zou L, Zhang J, Dou L. The comparison and application of corner detection algorithms. *J Multimed*. 2009;4(6):435–41.
- [3] Nguyen TP, Debled-Rennesson I. A discrete geometry approach for dominant point detection. *Pattern Recognit*. 2011;44(1):32–44.
- [4] Zhong B, Liao W. Direct curvature scale space: theory and corner detection. *IEEE Trans Pattern Anal Mach Intell*. 2007;29(3):508–12.



- [5] Zhang X, Song J, Dai G, Lyu MR. Extraction of line segments and circular arcs from freehand strokes based on segmental homogeneity features. *IEEE Trans Syst, Man, Cybern, Part B (Cybern)*. 2006;36(2):300–11. doi: 10.1109/tsmcb.2005.857288.
- [6] Mehrotra R, Nichani S, Ranganathan N. Corner detection. *Pattern Recognit*. 1990;23(11):1223–33.
- [7] Costagliola G, De Rosa M, Fuccella V. RankFrag: a machine learning-based technique for finding corners in hand-drawn digital curves. *21st International Conference on Distributed Multimedia Systems, DMS 2015*. doi: 10.18293/VLSS2015-043.
- [8] Wolin A, Eoff B, Hammond T. ShortStraw: a simple and effective corner finder for polylines. *SBIM*. 2008;8:33–40.
- [9] Xiong Y, LaViola JJ Jr. A shortstraw-based algorithm for corner finding in sketch-based interfaces. *Comput Graph*. 2010;34(5): 513–27.
- [10] Albert F, Fernández-Pacheco D, Aleixos N. New method to find corner and tangent vertices in sketches using parametric cubic curves approximation. *Pattern Recognit*. 2013;46(5):1433–48.
- [11] Albert F, Aleixos N. Improvements to the TCVD method to segment hand-drawn sketches. *Pattern Recognit*. 2017;63:416–26.
- [12] Herold J, Stahovich TF. Classyseg: a machine learning approach to automatic stroke segmentation. *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, 2011.
- [13] Long Z. *Robust corner and tangent point detection for strokes with deep learning approach*. arXiv preprint arXiv:1903.00899, 2019.
- [14] Staudemeyer RC, Morris ER. *Understanding LSTM—a tutorial into long short-term memory recurrent neural networks*. arXiv preprint arXiv:1909.09586, 2019.
- [15] Sherstinsky A. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenom*. 2020;404:132306.
- [16] Huang Z, Xu W, Yu K. *Bidirectional LSTM-CRF models for sequence tagging*. arXiv preprint arXiv:1508.01991, 2015.
- [17] Culjak I, Abram D, Pribanic T, Dzapo H, Cifrek M. A brief introduction to OpenCV. *2012 Proceedings of the 35th International Convention MIPRO*, pp. 1725–30, Opatija, Croatia, 2012.